Práctica 2.2. Sistema de Ficheros

Objetivos

En esta práctica se revisan las funciones del sistema básicas para manejar un sistema de ficheros, referentes a la creación de ficheros y directorios, duplicación de descriptores, obtención de información de ficheros o el uso de cerrojos.

Contenidos

Preparación del entorno para la práctica

La realización de esta práctica únicamente requiere del entorno de desarrollo (compilador, editores y utilidades de depuración). Estas herramientas están disponibles en las máquinas virtuales de la asignatura y en la máquina física de los puestos del laboratorio.

Creación y atributos de ficheros

El inodo de un fichero guarda diferentes atributos de éste, como por ejemplo el propietario, permisos de acceso, tamaño o los tiempos de acceso, modificación y creación. En esta sección veremos las llamadas al sistema más importantes para consultar y fijar estos atributos así como las herramientas del sistema para su gestión.

Ejercicio 1. `ls(1)` muestra el contenido de directorios y los atributos básicos de los ficheros. Consultar la página de manual y estudiar el uso de las opciones `-a -l -d -h -i -R -1 -F` y `--color`. Estudiar el significado de la salida en cada caso.

```
man 1 ls
```

Ejercicio 2. El *modo* de un fichero es `<tipo><rwx_propietario><rwx_grupo><rwx_resto>`:

- `tipo`: `-` fichero ordinario; `d` directorio; `l` enlace; `c` dispositivo carácter; `b` dispositivo bloque; `p` FIFO; `s` socket
- `rwx`: `r` lectura (4); `w` escritura (2); `x` ejecución (1)

Comprobar los permisos de algunos directorios (con `ls -ld`).

```
ls -ld
drwxr-xr-x 2 cursoredes cursoredes 116 Sep  9  2018
```

Ejercicio 3. Los permisos se pueden otorgar de forma selectiva usando la notación octal o la simbólica. Ejemplo, probar las siguientes órdenes (equivalentes):

- `chmod 540 fichero`
- `chmod u+rx,g+r-wx,o-wxr fichero`

¿Cómo se podrían fijar los permisos `rw-r--r-x`, de las dos formas?

```
-r-xr----- 1 cursoredes cursoredes 0 Nov  8 11:55 fichero
-r-xr----- 1 cursoredes cursoredes 0 Nov  8 11:55 ficheroo
```

Ejercicio 4. Crear un directorio y quitar los permisos de ejecución para usuario, grupo y otros. Intentar cambiar al directorio.

```
chmod 000 directorio_sin_permisos
cd directorio_sin_permisos
bash: cd: directorio_sin_permisos: Permission denied
```

Ejercicio 5. Escribir un programa que, usando `open(2)`, cree un fichero con los permisos `rw-r--r-x`. Comprobar el resultado y las características del fichero con `ls(1)`.

```
#include <stdio.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int main(){
    char path[]="/home/cursoredes/Downloads/jorge/fichero";
    int fd = open(path,O_CREAT,0645);
    return 1;
}

Ls -l
-rw-r--r-x 1 cursoredes cursoredes    0 Nov  8 12:31 fichero
```

Ejercicio 6. Cuando se crea un fichero, los permisos por defecto se derivan de la máscara de usuario (*umask*). El comando interno de la *shell* `umask` permite consultar y fijar esta máscara. Usando este comando, fijar la máscara de forma que los nuevos ficheros no tengan permiso de escritura para el grupo y no tengan ningún permiso para otros. Comprobar el funcionamiento con `touch(1)`, `mkdir(1)` y `ls(1)`.

```
umask 0027

Touch f
-rw-r----- 1 cursoredes cursoredes    0 Nov  8 12:37 f
Mkdir carpeta
drwxr-x--- 2 cursoredes cursoredes    6 Nov  8 12:36 carpeta
```

Ejercicio 7. Modificar el ejercicio 5 para que, antes de crear el fichero, se fije la máscara igual que en el ejercicio 6. Comprobar el resultado con `ls(1)`. Comprobar que la máscara del proceso padre (la *shell*) no cambia.

```
int main(){
    umask(0027);
    char path[]="/home/cursoredes/Downloads/jorge/ej7";
    int fd = open(path,O_CREAT,0645);
    return 1;
}
```

```
umask
0022
./a.out
umask
0022

-rw-r----- 1 cursoredes cursoredes    0 Nov  8 12:31 ej7
```

Ejercicio 8. `ls(1)` puede mostrar el inodo con la opción `-i`. El resto de información del inodo puede obtenerse usando `stat(1)`. Consultar las opciones del comando y comprobar su funcionamiento.

```
ls -i -1
53013172 a.out
   30819 dir
51269384 ej7
53013169 fachero
53013173 fichero
51269383 f
53013165 fuchero
53013168 pr.c
53013174 carpeta

stat ej7
  File: ‘ej7’
  Size: 0           Blocks: 0          IO Block: 4096   regular empty file
Device: fd00h/64768d      Inode: 51269384    Links: 1
Access: (0640/-rw-r-----)  Uid: ( 1000/cursoredes)   Gid: ( 1000/cursoredes)
Access: 2021-11-08 12:43:42.310684040 +0100
Modify: 2021-11-08 12:43:42.310684040 +0100
Change: 2021-11-08 12:43:42.310684040 +0100
 Birth: -
```

Ejercicio 9. Escribir un programa que emule el comportamiento de `stat(1)` y muestre:

- El número *major* y *minor* asociado al dispositivo.
- El número de inodo del fichero.
- El tipo de fichero (directorio, enlace simbólico o fichero ordinario).
- La hora en la que se accedió el fichero por última vez. ¿Qué diferencia hay entre `st_mtime` y `st_ctime`?

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/sysmacros.h>
#include <time.h>

int main(int argc, char **argv) {
      if (argc < 2) {
              printf("error parametros\n");
              return -1;
```

```
	}
	struct stat buff;
	int stat_= stat(argv[1], &buff);
	if (stat_ == -1) {
		printf("error directorio\n");
		return -1;
	}
	printf("major: %li\n", (long) minor(buff.st_dev));
	printf("minor: %li\n", (long) major(buff.st_dev));
	printf("inodo: %li\n",buff.st_ino);
	printf("tipo: %i\n",buff.st_mode);
	mode_t mode = buff.st_mode;
	if (S_ISLNK(mode)){
		printf("enlace simbólico\n");
	} else if (S_ISREG(mode)) {
		printf("regular\n");
	} else if (S_ISDIR(mode)) {
		printf("directorio\n");
	}
	time_t t = buff.st_atime;
	struct tm *tm= localtime(&t);
	printf("acceso: %d:%d\n", tm->tm_hour, tm->tm_min);
	return 0;
}

./a.out carpeta
major: 0
minor: 253
inodo: 18833075
tipo: 16872
directorio
acceso: 20:48

La diferencia entrest_mtime y st_ctime es que M considera modificaciones internas del archivo y C considera cualquier modificación que corresponda al archivo: permisos,etc
```

Ejercicio 10. Los enlaces se crean con `ln(1)`:

- Con la opción `-s`, se crea un enlace simbólico. Crear un enlace simbólico a un fichero ordinario y otro a un directorio. Comprobar el resultado con `ls -l` y `ls -i`. Determinar el inodo de cada fichero.
- Repetir el apartado anterior con enlaces rígidos. Determinar los inodos de los ficheros y las propiedades con `stat` (observar el atributo número de enlaces).
- ¿Qué sucede cuando se borra uno de los enlaces rígidos? ¿Qué sucede si se borra uno de los enlaces simbólicos? ¿Y si se borra el fichero original?

```
touch fiord
mkdir dir
ln -s fiord fiordsb
ln -s dir dirsb
```

```
ls -li
total 16
 3577020 -rwxr-xr-x 1 cursoredes cursoredes 8768 Nov 13 21:18 a.out
18833075 drwxr-x--- 2 cursoredes cursoredes    6 Nov 13 20:48 carpeta
33974116 drwxr-x--- 2 cursoredes cursoredes    6 Nov 13 21:23 dir
  309735 lrwxrwxrwx 1 cursoredes cursoredes    3 Nov 13 21:24 dirsb -> dir
  309732 -rw-r----- 1 cursoredes cursoredes    0 Nov 13 20:59 ej7
  309733 -rw-r----- 1 cursoredes cursoredes    0 Nov 13 21:23 fiord
  309734 lrwxrwxrwx 1 cursoredes cursoredes    5 Nov 13 21:24 fiordsb -> fiord
 3577023 -rw-rw-r-- 1 cursoredes cursoredes  872 Nov 13 21:17 pr.c

touch fi
mkdir direc
ln fi fidu
ln direc direcdu
ln: ‘direc’: hard link not allowed for directory
ls -li

309736 -rw-r----- 2 cursoredes cursoredes    0 Nov 13 21:29 fi
309736 -rw-r----- 2 cursoredes cursoredes    0 Nov 13 21:29 fidu
Mismo inodo


rm -rf fidu
rm -rf fiordsb
ln -s fiord fiordsb
rm -rf fiord
309734 lrwxrwxrwx 1 cursoredes cursoredes    5 Nov 13 21:35 fiordsb -> fiord
Se corrompe
```

Ejercicio 11. `link(2)` y `symlink(2)` crean enlaces rígidos y simbólicos, respectivamente. Escribir un programa que reciba una ruta a un fichero como argumento. Si la ruta es un fichero regular, creará un enlace simbólico y rígido con el mismo nombre terminado en `.sym` y `.hard`, respectivamente. Comprobar el resultado con `ls(1)`.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/sysmacros.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char **argv) {
	if (argc < 2) {
		printf("error parametros\n");
		return -1;
	}
	struct stat buff;
```

```
	int stat_= stat(argv[1], &buff);
	if (stat_ == -1) {
		printf("error directorio\n");
		return -1;
	}
	char* duro = malloc(sizeof(char)*(5 + strlen(argv[1])));
	char* simb = malloc(sizeof(char)*(5 + strlen(argv[1])));
	strcpy(duro, argv[1]);
	strcpy(simb, argv[1]);
	duro = strcat(duro, ".hard");
	simb = strcat(simb, ".sym");
	printf("duro: %s\n", duro);
	printf("simbolico: %s\n", simb);
	mode_t stmode = buff.st_mode;
	if (S_ISREG(stmode)) {
		printf("%s es regular\n", argv[1]);
		if (link(argv[1],duro) == -1) {
			 printf("error enlace duro\n");
		}
		else
			printf("enlace duro\n");
		if (symlink(argv[1],simb) == -1) {
			printf("error enlace simbolico\n");
		}
		else
			printf("enlace simbolico\n");
	}
	 else  {
		printf("error regular\n");
	}
	return 0;
}

./a.out f
duro: f.hard
simbolico: f.sym
f es regular
enlace duro
enlace simbolico

309737 -rw-r--r-- 2 cursoredes cursoredes    0 Nov 13 21:53 f
309737 -rw-r--r-- 2 cursoredes cursoredes    0 Nov 13 21:53 f.hard
309738 lrwxrwxrwx 1 cursoredes cursoredes    1 Nov 13 21:54 f.sym -> f
```

Redirecciones y duplicación de descriptores

La *shell* proporciona operadores (>, >&, >>) que permiten redirigir un fichero a otro, ver los ejercicios

propuestos en la práctica opcional. Esta funcionalidad se implementa mediante `dup(2)` y `dup2(2)`.

Ejercicio 12. Escribir un programa que redirija la salida estándar a un fichero cuya ruta se pasa como primer argumento. Probar haciendo que el programa escriba varias cadenas en la salida estándar.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
int main(int argc, char **argv) {

      if (argc < 2) {
             printf("error parametros\n");
             return -1;
      }
  int fd = open(argv[1], O_CREAT | O_RDWR, 00777);
  if (fd == -1) {
    printf("error abrir fichero\n");
    return -1;
  }
  int fd_dup = dup2(fd, 1);
  printf("hemos redirigido la salida estandar a %s\n", argv[1]);
  dup2(fd_dup, fd);
  return 1;
}

gcc pr.c
./a.out ej12
cat ej12
hemos redirigido la salida estandar a ej12
```

Ejercicio 13. Modificar el programa anterior para que también redirija la salida estándar de error al fichero. Comprobar el funcionamiento incluyendo varias sentencias que impriman en ambos flujos. ¿Hay diferencia si las redirecciones se hacen en diferente orden? ¿Por qué `ls > dirlist 2>&1` es diferente a `ls 2>&1 > dirlist`?

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
int main(int argc, char **argv) {
      if (argc < 2) {
             printf("error parametros\n");
             return -1;
      }
    int fd = open(argv[1], O_CREAT | O_RDWR, 00777);
    if (fd == -1) {
        printf("error abrir fichero\n");
        return -1;
    }
```

```
    int fd_dup_2 = dup2(fd,2);
    int fd_dup = dup2(fd, 1);
    printf("hemos redirigido la salida estandar a %s\n", argv[1]);
    char *s;
      if (setuid(0) == -1){
             perror(s);
      }
    dup2(fd_dup, fd);
    dup2(fd_dup_2, fd);
    return 1;
}

gcc pr.c
./a.out ej13
cat ej13
1◆I◆◆^H◆◆H◆◆◆PTI◆◆◆@: Operation not permitted
hemos redirigido la salida estandar a ej13


ls > dirlist 2>&1 es diferente a ls 2>&1 > dirlist?
El primero redirecciona: estandar -> dirlist y errores -> 1=dirlist
El segundo redirecciona: errores -> estandar y estandar -> dirlist
```

Cerrojos de ficheros

El sistema de ficheros ofrece cerrojos de ficheros consultivos.

Ejercicio 14. El estado y cerrojos de fichero en uso en el sistema se pueden consultar en el fichero `/proc/locks`. Estudiar el contenido de este fichero.

```
cat /proc/locks
1: POSIX  ADVISORY  WRITE 1596 fd:00:53013121 0 EOF
2: POSIX  ADVISORY  WRITE 1589 fd:00:53013120 0 EOF
3: POSIX  ADVISORY  WRITE 1582 fd:00:53012991 0 EOF
4: POSIX  ADVISORY  WRITE 1572 fd:00:53012987 0 EOF
5: POSIX  ADVISORY  WRITE 1367 00:13:21418 0 EOF
6: FLOCK  ADVISORY  WRITE 1329 fd:00:17458434 0 EOF
7: FLOCK  ADVISORY  WRITE 1329 fd:00:30831 0 EOF
8: FLOCK  ADVISORY  WRITE 1074 00:13:20107 0 EOF
9: POSIX  ADVISORY  WRITE 1072 00:13:20016 0 EOF
10: POSIX  ADVISORY  WRITE 822 00:13:17996 0 EOF
12: POSIX  ADVISORY  WRITE 481 00:13:12689 0 EOF

Id : Tipo(P/F) Modo Tipo(W/R) PID idfichero 1ºbit_bloqueado último_bit_bloqueado
```

Ejercicio 15. Escribir un programa que consulte y muestre en pantalla el estado del cerrojo sobre un fichero usando `lockf(3)`. El programa mostrará el estado del cerrojo (bloqueado o desbloqueado). Además:

- Si está desbloqueado, fijará un cerrojo y escribirá la hora actual. Después suspenderá su ejecución durante 30 segundos (con `sleep(3)`) y a continuación liberará el cerrojo.
- Si está bloqueado, terminará el programa.

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <time.h>
#include <string.h>
int main(int argc, char ** argv) {
    if (argc < 2) {
        printf("error parametros\n");
        return -1;
    }
    int fd = open(argv[1], O_CREAT | O_RDWR, 00777);
    if (fd == -1) {
        printf("error abrir fichero\n");
        return -1;
    }
    struct flock lock;
    lock.l_type = F_UNLCK;
    lock.l_whence = SEEK_SET;
    lock.l_pid = getpid();
    lock.l_start = 0;
    lock.l_len = 0;
    int st = fcntl(fd, F_GETLK, &lock);
    if (lock.l_type == F_UNLCK) {
        printf("lock desbloqueado, fija cerrojo y escribe hora actual\n");
        lock.l_type = F_WRLCK;
        lock.l_whence = SEEK_SET;
        lock.l_pid = getpid();
        lock.l_start = 0;
        lock.l_len = 0;
        if (fcntl(fd, F_GETLK, &lock) == -1) {
            printf("error crear lock\n");
            close(fd);
            return 1;
        }
        else {
            printf("lock escritura\n");
            time_t t = time(NULL);
            struct tm *tm = localtime(&t);
            char buff[1024];
            sprintf (buffer, "Hora Actual: %d:%d\n", tm->tm_hour, tm->tm_min);
            write(fd, &buff, strlen(buff));

            sleep(30);

            lock.l_type = F_WRLCK;
            lock.l_whence = SEEK_SET;
            lock.l_start = 0;
            lock.l_len = 0;
            lock.l_pid = getpid();
            if (fcntl(fd, F_GETLK, &lock) == -1) {
                printf("error crear lock\n");
```

```
                close(fd);
                return 1;
            }
            else
                close(fd);
        }
    }
    else {
        printf("lock bloquedado, termina el programa \n");
        close(fd);
        return 1;
    }
    close(fd);
}
gcc pr.c
./a.out ej15
lock desbloqueado, fija cerrojo y escribe hora actual
lock escritura
cat ej15
Hora Actual: 0:20simbolico
```

***Ejercicio 16* (Opcional).** `flock(1)` proporciona funcionalidad de cerrojos antiguos BSD en guiones *shell*. Consultar la página de manual y el funcionamiento del comando.

```
man flock : flock - manage locks from shell scripts
```

Directorios

Ejercicio 17. Escribir un programa que cumpla las siguientes especificaciones:

- El programa tiene un único argumento que es la ruta a un directorio. El programa debe comprobar la corrección del argumento.
- El programa recorrerá las entradas del directorio de forma que:
    - Si es un fichero normal, escribirá el nombre.
    - Si es un directorio, escribirá el nombre seguido del carácter '/'.
    - Si es un enlace simbólico, escribirá su nombre seguido de '->' y el nombre del fichero enlazado. Usar `readlink(2)` y dimensionar adecuadamente el *buffer*.
    - Si el fichero es ejecutable, escribirá el nombre seguido del carácter '*'.
- Al final de la lista el programa escribirá el tamaño total que ocupan los ficheros (no directorios) en kilobytes.

```
#include <sys/types.h>
#include <dirent.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
int main(int argc, char **argv) {
      if (argc < 2) {
            printf("error parametros\n");
```

```
            return -1;
      }
    DIR *dir = opendir(argv[1]);
    if (dir == NULL ) {
        printf("error directorio\n");
        return -1;
    }
    struct dirent *actual;
    struct stat info;
    unsigned long int tamano = 0;
    actual = readdir(dir);
    size_t path_tam = strlen(argv[1]);
    while (actual != NULL) {
        char *path = malloc(sizeof(char)*(3+path_tam+strlen(actual->d_name)));
        strcpy(path, argv[1]);
        strcat(path, "/");
        strcat(path, actual->d_name);
        if (stat(path, &info) == -1) {
            printf("error archivo\n");
            free(path);
            closedir(dir);
            return -1;
        }
        else {
            if (S_ISLNK(info.st_mode)) {
                char *fich_enlazado = malloc(info.st_size + 1);
                int rc2 = readlink(path, fich_enlazado, info.st_size + 1);
                printf("simbolico %s -> %s \n", actual->d_name, fich_enlazado);
                free(fich_enlazado);
              }
            else if (S_ISREG(info.st_mode)) {
                printf("Normal %s \n", actual->d_name);
                tamano = tamano + ((info.st_blksize/8)*info.st_blocks);
            }
            else if (S_ISDIR(info.st_mode)) {
                printf("Directorio %s\\ \n", actual->d_name);
            }
        }
        free(path);
        actual = readdir(dir);
    }
    printf("Tamaño que ocupan los ficheros: %i \n",tamano) ;
    closedir(dir);
    return 0;
}

gcc pr.c
./a.out ../jorge
Directorio .\
Directorio ..\
Directorio carpeta\
```

```
Normal pr.c
Normal ej7
Directorio dir\
simbolico dirsb -> dir
Normal fi
Directorio direc\
Normal prc.c
Normal f
Normal f.hard
simbolico f.sym-> f
Normal ej12
Normal ej13
Normal ej15
Normal a.out
Tamaño que ocupan los ficheros: 36864
```